## TEMA 1: SELECT

* Uso de operadores aritméticos: se pueden utilizar en la sentencia select.
* Ejemplo: SELECT DNI, SALARIO +300 FROM ASALARIADOS;

El orden de prioridad de los operadores es \* / + - . Si tienen la misma prioridad se evalúan de izquierda a derecha.

* Valores NULL: Cuando un campo no tiene ningún valor. Distinto de 0. Por ejemplo, si divido algo por un campo NULL, el resultado será desconocido o NULL, pero si divido algo entre 0, saldrá error.
* ALIAS: Para darle nombres más descriptivos a las columnas. Van asociados a la palabra AS, pero se pueden crear sin utilizarla. Necesario poner comillas dobles o sin ellas.

Ejemplo: SELECT SALARIO AS “Sueldo” FROM ASALARIADO;

Ejemplo: SELECT SALARIO “Sueldo” FROM ASALARIADO;

Ejemplo: SELECT SALARIO AS Sueldo FROM ASALARIADO

## TEMA 2: WHERE

* **DESCRIBE** o **DESC**: para mostrar la estructura de una tabla
* DESC ASALARIADO o DESCRIBE ASALRIADO
* **CONCATENACION**: sirve para concatenar las columnas. Se hace con **||** y se puede concatenar los caracteres que queramos. Además, también se les puede dar una alias.
* Ejemplo: SELECT DNI||” “||SALARIO FROM ASALARIADO.
* Ejemplo: SELECT ID||” “||INFO AS “Departament info” FROM DEPARTAMENTO

También se pueden poner valores literales mediante comillas simples y crear frases.

* Ejemplo: SELECT 'EL EMPLEADO '||DNI||' TIENE UN SALARIO DE '||SALARIO AS SALARIO FROM ASALARIADO;
* **DISTINCT**: sirve para eliminar las filas duplicadas en la sentencia SELECT. No las elimina de la tabla.
* Ej. SELECT DISTINCT DEPARTAMENT\_ID FROM EMPLEADOS;
* **WHERE:** Para limitar el número de filas que devuelve SELECT. Las fechas y caracteres que se utilicen debe estar entre comillas simples mientras que los números no. Los operadores que se pueden utilizar son: =,>,<,<=,>=,<> no es igual que(o!= o ^=).
* Ej. SELECT \* FROM EMPLEADOS WHERE SALARIO >300;
* EJ. SELECT \* FROM ASALARIADO WHERE SALARIO <> 678;
* EJ. SELECT \* FROM DONACION WHERE FECHA< ‘12/03/2001’;
* **OPERADORES DE COMPARACIÓN: BETWEEN…AND, IN, LIKE**
* Ej. SELECT \* FROM DONACION WHERE FECHA **BETWEEN** ‘01/06/20’ **AND** ‘30/06/20’; 🡪Equivalente a utilizar > … AND <…
* EJ. SELECT \* FROM TRABAJADOR WHERE NOMBRE **IN** (‘MARIO GARCÍA’, ‘ELENA SUAREZ’); 🡪 Devuelve todos los trabajadores cuyo nombre sea alguno de esos dos. Equivalente a utilizar sentencias OR.
* Ej. SELECT NOMBRE FROM TRABAJADOR WHERE NOMBRE LIKE ‘\_o%’🡪 selecciona todos los nombre cuya segunda letra sea o. se pueden utilizar dos caracteres comodín: “\_” que representa un único carácter y “%” que representa una secuencia de cero o más caracteres. Si no queremos que estos sean los caracteres comodín, los podemos cambiar con **ESCAPE.**
* Ej. SELECT \* FROM TRABAJADOR WHERE NOMBRE LIKE ‘%/\_R%’ ESCAPE ‘/’; 🡪 estamos buscando nombres con la secuencia ‘\_R’ y no solo ‘R’.
* Ej. SELECT NOMBRE FROM TRABAJADOR WHERE NOMBRE **IS NULL**; 🡪 Estamos buscando los trabajadores cuyo campo nombre sea desconocido, sin asignar o inaccesibles.

## TEMA 3: FUNCIONES

### OPERADORES LÓGICOS: AND, OR, NOT

* AND: deben cumplirse todas las condiciones para que devuelva TRUE
* OR: debe cumplirse alguna de las condiciones
* NOT: no debe cumplirse ninguna condición para que devuelva TRUE

EJ: SELECT last\_name||' '||salary\*1.05 As "Employee Raise" FROM employees WHERE department\_id IN(50,80) AND first\_name LIKE 'C%' OR last\_name LIKE '%s%';

El orden de evaluación que sigue SQL es: (Este orden se puede modificar utilizando paréntesis)



### ORDENACIÓN DE FILAS

Para ordenar las filas seleccionadas con la sentencia select utilizamos la sentencia ORDER BY. Podemos ordenar en orden ascendente con ASC o descendente con DESC. Podemos elegir también cuando deben salir los valores NULL: al principio con NULLS FIRST o al final con NULLS LAST. Se pueden ordenar en base a más de una columna, para ello separamos las columnas con comas. Además, la cláusula ORDER BY debe ser la última cláusula de la sentencia SQL. También podemos usar alias y ordenar en base a ellos. Se puede ordenar en base a una columna que no hayamos seleccionado en la sentencia SELECT.

* Ej. SELECT \* FROM TRABAJADOR ORDER BY SALARIO ASC NULLS FIRST🡪 Muestra los trabajadores ordenados por salario en orden ascendente y con los NULLS al principio.
* Ej. SELECT Nombre, FUNCION AS ‘trabajo’ FROM TRABAJADORES ORDER BY ‘trabajo’;
* Ej. SELECT Nombre FROM TRABAJADOR ORBER BY SALARIO;
* Ej. SELECT Nombre FROM TRABAJDOR ORDER BY FUNCION ASC, NOMBRE;

### FUNCIONES

* Funciones de una fila: aceptan uno o varios argumentos y devuelven uno por fila. Por lo tanto, si se aplica una función de una sola fila a 12 filas, se obtendrá 12 resultados de la función de una sola fila.
* Funciones de varias filas: toman muchas filas como entrada y devuelven un único valor como salida. La entrada de filas puede ser toda la tabla o la tabla dividida en grupos más pequeños. Por ejemplo:
* MAX: Busca el valor máximo en un grupo de filas
* MIN: Busca el valor mínimo en un grupo de filas
* AVG: Busca el valor promedio de un grupo de filas.

### MANIPULACIÓN DE MAYUSCULAS/MINUSCULAS Y CARACTERES

* TABLA **DUAL**: Para crear sentencias select que no están relacionadas con ninguna tabla de la base de datos. Tiene una fila X y una columna DUMMY.
* Ej. SELECT (319/29)+12 FROM DUAL; el valor devuelto es el cálculo del valor ejecutado.
* Manipulación de mayúsculas/minúsculas: se pueden utilizar en el **SELECT**, **WHERE** u **ORDER** **BY**.
* Lower: convierte los caracteres en minúsculas.

Ej. Select \* from trabajador where LOWER(nombre)=’mario garcia’: busca el trabajador cuyo nombre es minúsculas es ese

* Upper: convierte los caracteres en mayúsculas

Ej. Select UPPER(nombre) from trabajadores: devuelve todos los trabajadores con sus nombres en mayúsculas

* INITCAP: Convierte la inicial de cada palabra en mayúscula

Ej. Select \* from trabajador where INITCAP(nombre)=’Mario Garcia’: convierte la inicial de cada palabra en mayúscula.

* Funciones de manipulación de caracteres:
* **CONCAT**: Une dos valores. Es el equivalente a || pero solo se pueden unir dos valores.

Ej. Select CONCAT(nombre, DNI) from trabajador;

Ej. Select CONCAT(‘HELLO’,’WORLD’) FROM DUAL;

* **SUBTR**: Extrae una cadena de una longitud determinada. Los argumentos son la cadena, la posición inicial y la longitud

Ej. Select SUBTR(‘HELLO’,2,3) FROM DUAL; 🡪 devuelve ELL

Ej. Select SUBTR(Nombre,1,3) from trabajador

* **LENGTH**: Muestra la longitud de una cadena con un valor numérico.

Ej. SELECT LENGTH(Nombre) from trabajador;

Ej. Select Length(‘HELLO’( from dual;

* **INSTR**: Encuentra la posición numérica de los caracteres especificados. Si no se encuentra devuelve 0.

Ej. SELECT INSTR(‘HELLO’,’H’)FROM DUAL;🡪Devuelve 1

Ej. Select INSTR(Nombre, ‘ar’) from trabajador

* **LAPD**: Rellena la parte izquierda de una cadena con una serie de caracteres. Los argumentos son la cadena, el número total de caracteres y el carácter con la que rellenarla.

Ej. Select LPAD(‘Nombre’,5,’\*’) from trabajador;🡪 devuelve \*\*\*\*\*mario garcia

* **RPAD**: Igual que LPAD pero en la parte derecha
* **TRIM**: Elimina todos los caracteres especificados, ya sea del principio(LEADING), del final(TRAILING) o de ambos(BOTH).

Ej. SELECT TRIM(BOTH ‘M’ from nombre) from trabajador🡪 devuelve ario Gomez

Ej. Select TRIM(LEADING ‘H’ from ‘Hello’) from deal🡪 devuelve ello.

* **REPLACE**: Sustituye una secuencia de caracteres de una cadena por otra secuencia. Los argumentos son la cadena inicial, la cadena que se debe reemplazar y la cadena que se pondrá por la reemplazada.

Ej. Select REPLACE(‘JACK’,’AC’,’OR’) FROM DEAL;🡪Devuelve JORK

EJ. Select REPLACE(Nombre, ‘ar’, ‘or’) from trabajador;

* Funciones numéricas: TRUNC, ROUND Y MOD
* **ROUND**: se usa tanto con números como con fechas. Se usa para redondear números. Sus argumentos son la columna o expresión y el número de decimales con a los que queremos redondear.

Ej. ROUND(45.967,2)🡪45.97

Ej. ROUND(45.975)🡪46

Ej. ROUND(45.975,-1)🡪50

* **TRUNC**: Se usa tanto con números como con fechas. Se usa utilizar un numero especificado de decimales

Ej. TRUNC(46.978,1)🡪46.9

Ej. TRUNC(46.978,0)🡪46

* **MOD**: Encuentra el resto de una división. Se puede utilizar para saber si un número es par o impar si se hace entre 2. Si devuelve 0 es par y si devuelve 1 es impar.

Ej. Select nombre, MOD(SALARIO,2) AS RESTO FROM TRABAJADOR;

* Funciones de fechas:
* SYSDATE: función que devuelve la fecha y hora actual del servidor.
* Trabajar con fechas:

Ej. Select id\_donacion, fecha+60 from donacion;🡪 muestra el id de la donacion y la fecha más 60 días.

Ej. Select id\_donacion, (SYSDATE-fecha)/7 from donacion;🡪 número de semanas desde que se hizo la donación

Ej. Si se divide entre 360 sale el número de años.

FUNCIONES DE FECHAS:

* MONTHS\_BETWEEN: Numero de meses entre dos fechas.

Ej. select id\_donacion, fecha from donacion where months\_between (SYSDATE,fecha)>5;🡪 muestra las donaciones de hace más de 5 semanas

* ADD\_MONTHS: Suma un numero de meses. Si el argumento es negativo, se restarán

Ej. Select ADD\_MONTHS(SYSDATE,12) AS NEXT YEAR FROM DUAL

* NEXT\_DAY: Fecha del día de la siguiente incidencia de ese día de la semana especificado.

Ej. Select NEXT\_DAY(SYSDATE,”Saturday”) from dual🡪 devuelve la fecha del siguiente sábado

* LAST\_DAY: Toma una fecha y devuelve el ultimo día de ese mes

Ej. Select LAST\_DAY(SYSDATE) FROM DUAL🡪 31/05/21

* ROUND: Redondea la fecha seleccionada a la unidad especificada.

Ej. Select ROUND(SYSDATE,’Month’) FROM DUAL; 🡪 redondea al mes que este más cerca

* TRUNC: Devuelve fecha truncada a la unidad que le hayamos especificado

Ej. SELECT TRUNC(SYSDATE,’Month’) FROM DUAL🡪 Devuelve el mes al que pertenece esa fecha.

## TEMA 4: JOIN

### Uniones cruzadas y uniones naturales

* Unión natural(**NATURAL JOIN**): une todas las columnas de dos o más tablas en el caso de que alguna de ellas tenga alguna columna con el mismo nombre. No hace falta especificar las columnas que queremos mostrar porque se mostraran todas. En el where se pueden utilizar columnas de cualquiera de las tablas.
* Ej. Select \* from TRABAJADOR NATURAL JOIN VOLUNTARIO WHERE NUMAYUDAS>5;🡪muestra toda la información de los trabajadores-voluntarios con ayudas >5.
* **CROSS JOIN:** Une cada fila de una tabla a cada fila de otra tabla. Esto da resultados muy grandes porque prueba todas las posibles combinaciones.

### Clausulas Join:

* **USING:** especifica las columnas que se deben utilizar para la unión ya que con la natural si hay columnas con el mismo nombre, pero diferente tipo de dato saltará un error. Las columnas a las que se hace referencia en la clausula USING no deben tener un cualificador(alias o nombre) y deben ir entre paréntesis. La sintaxis es JOIN…USING…
* ej. select nombre,dni, numayudas from trabajador join voluntario using (dni) where numayudas>5;🡪 solo se compara la columna dni de las dos tablas.
* **Alias de tabla**: se utiliza para diferencia a que tabla pertenece la columna a la que nos referimos cuando estamos trabajando con columnas que tienen el mismo nombre en varias tablas. Cuando los nombres de columna no estén duplicados, no es necesario utilizar alias. Tanto en la cláusula select como en el from o en el where se debe utilizar el mismo alias en caso de que lo hayamos utilizado
* Select t.nombre, t.dni from trabajador t, asalariado a where t.dni=a.dni;
* Select trabajador.nombre, trabajador.dni from trabajador, asalariado where trabajador.dni=asalariado.dni;🡪 igual que la anterior, pero sin alias
* **ON:** Cuando las columnas comunes tengan diferente nombre o se utilice algún operados no podemos utilizar USING. En estos casos usaremos ON. Si los nombres son distintos no hará falta utilizar alias de tabla, pero si son iguales habrá que usar alias porque si no dará error. También se pueden unir dos tablas que no tengan ninguna columna en común(podríamos utilizar operadores o clausulas como BETWEEN, LIKE)
* Select t.nombre,t.dni from trabajador t Join asalariado a ON t.dni=a.dni
* Select matricula, id\_donacion from donacion Join vehículo ON matricula=vehículo where destino like ‘Valencia’;🡪 al tener las columnas nombres distintos, no es necesario usar alias.
* Select matricula, id\_donacion from donacion Join vehículo ON(matricula=vehículo AND destino like ‘Valencia’); 🡪 igual que la anterior, pero sin usar where y metiéndolo todo en la cláusula ON.
* Unir **3 tablas** o más:
* select t.nombre,t.dni, d.destino, v.matricula from trabajador t join donacion d on (t.dni=d.trabajador) join vehiculo v on (d.vehiculo= v.matricula) where d.destino like 'Valencia';

### Uniones internas y externas:

Una unión de dos o más tablas que devuelven solo las filas coincidentes se llama unión externa mientras que cuando devuelven las filas coincidentes y las no coincidentes se llama unión externa.

* Unión externa izquierda(LEFT OUTER JOIN): La tabla que aparezca antes de left outer Join se le llama tabla izquierda y saldrán todas sus filas coincidan o no coincidan.
* SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e LEFT OUTER JOIN departments d ON (e.department\_id = d.department\_id);🡪 muestra todos los apellidos aunque no tengan asignado un departamento
* Unión externa derecha(RIGTH OUTER JOIN): Igual que la anterior pero a la inversa. Se muestran todas las filas de la tabla derecha aunque no coincida con ninguna de la tabla izquierda
* SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e RIGHT OUTER JOIN departments d ON (e.department\_id = d.department\_id);
* Unión externa completa: se muestran las coincidentes y no coincidentes de ambas tablas. Une todas las filas de una unión izquierda y una unión derecha sin duplicación.
* SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e FULL OUTER JOIN departments d ON (e.department\_id = d.department\_id);

### Autouniones y consultas jerárquicas

* Autounión: se utiliza para unir una tabla a sí misma como si se tratara de dos tablas distintas. Para ello se le asignan dos nombres o alias. Esto se denomina SELF-JOIN.
* SELECT worker.last\_name || ' works for ' || manager.last\_name AS "Works for" FROM employees worker JOIN employees manager ON (worker.manager\_id = manager.employee\_id);

### Unión igualitaria y producto cartesiano de ORACLE

Repaso de todo lo anterior, pero con la sintaxis de Oracle. Aquí por ejemplo para unir varias tablas no es necesario poner tantos Join…on o Join…using; basta con poner algo así(esto solo sirve para uniones igualitarias o cross-join):

* SELECT last\_name, city FROM employees e, departments d, locations l WHERE e.department\_id = d.department\_id AND d.location\_id = l.location\_id;

### Uniones no igualitarias y uniones externas de ORACLE

Una unión no igualitaria es equivalente a ANSI JOIN ON donde la condición utilizada es algo distinto de =.

* SELECT last\_name, salary, grade\_level, lowest\_sal, highest\_sal FROM employees, job\_grades WHERE (salary BETWEEN lowest\_sal AND highest\_sal);

En las uniones externas, para indicar que tabla puede tener datos que faltan, indicar + después del nombre de la columna en la cláusula where.

* SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e, departments d WHERE e.department\_id = d.department\_id(+);🡪 equivalente a ANSI LEFT JOIN.
* SELECT e.last\_name, d.department\_id, d.department\_name FROM employees e, departments d WHERE e.department\_id(+) = d.department\_id;🡪 equivalente a ANSI RIGTH JOIN.

**No hay equivalente para FULL OUTER JOIN**

## TEMA 5: AGRUPAMIENTO

### Funciones de grupo

Las funciones de grupo se escriben en la cláusula select. No se pueden utilizar en la cláusula WHERE. Estas funciones ignoran los valores NULL.

Las funciones de grupo son **AVG, SUM, COUNT, MIN, MAX, VARIANCE, STDDEV**.

* select avg(numayudas)as media, max(numayudas)as maximo, min(numayudas)as minimo, variance(numayudas) as varianza, round(stddev(numayudas),2) as sd from voluntario;

### COUNT, DISTINCT,NVL

* **COUNT**: Devuelve el número de valores no nulos de la columna especificada. Count(\*) devuelve el numero de filas de la tabla, independientemente de que alguna columna tenga valores nulos.
* Select count(funcion) from trabajador;
* Select funcion, count(funcion) from trabajador group by funcion;🡪 agrupa la cuenta por funcion para mostrar el numero de trabajadores que desempeña cada funcion.
* **DISTINCT**: Se usa para devolver solo valores no duplicados. Se puede utilizar con todas las funciones de grupo.
* Select Distinct(funcion) from trabajador;🡪 muestra todas la funciones que hay
* Select count(Distinct(funcion)) from trabajador🡪 muestra el número total de funciones que hay.
* **NVL**: Sirve para tener en cuenta también los valores nulos. Los argumentos son la columna que se debe tratar y el valor que se le debe dar a los valores nulos.
* select AVG(NVL(customer\_orders,0)) from ORDERS;🡪Devuelve la media de demandas de los clientes teniendo en cuenta los que no han pedido nada.

### Uso de las cláusulas GROUP BY y HAVING

* GROUP BY: Para dividir las filas de una tabla en subgrupos.
* Select jornada, AVG(salario) from asalariado group by jornada;🡪 hace la media del salario agrupándolo por tipo de jornada

Recordad que las funciones de grupo ignoran los valores nulos. Además, si incluye una función de grupo(AVG,SUM,COUNT,MAX,MIN,STDDEV,VARIANCE) en una clausula SELECT junto con cualquier otra columna individual, cada columna individual también debe aparecer en la cláusula GROUP BY.

A veces hay que dividir esos grupos en subgrupos más pequeños(GRUPOS DENTRO DE GRUPOS)

* SELECT department\_id, job\_id, count(\*) FROM employees WHERE department\_id > 40 GROUP BY department\_id, job\_id; 🡪 divide los empleados por departamento y en cada apartamento los divide por trabajo.

Las funciones de grupo se pueden anidar en una profundidad de 2 cuando se utilice GROUP BY.

* Select MAX(AVG(SALARIO)) FROM asalariado group by jornada🡪 devuelve el salario medio máximo de los asalariados dividido por tipo de jornada.
* HAVING: Igual que se usa WHERE para restringir las filas que se seleccionan, se puede utilizar HAVING para restringir los grupos devueltos por una clausula GROUP BY. Por tanto, se pone después de esta cláusula.
* SELECT jornada, AVG(salario) from asalariado group by jornada HAVING COUNT(SALARIO)>2🡪 Muestra el sueldo medio por tipo de jornada siempre que haya mas de dos empleados con ese tipo de jornada.

### Uso de los operadores SET

Se van a utilizar dos tablas: A=1,2,3,4,5 y B=4,5,6,7,8. Ambas tablas es necesario que tengan el mismo numero de columnas y han de ser del mismo tipo.

* **UNION**: Devuelve todas las filas de ambas columnas después de eliminar los duplicados, es decir, los que están en las dos tablas.
* SELECT a\_id from A UNION SELECT b\_id from B;🡪 mostrará 1,2,3,4,5,6,7,8
* **UNION** **ALL**: Devuelve todas las filas de ambas tablas sin eliminar los duplicados.
* SELECT a\_id from A UNION ALL select b\_id from B;🡪 devuelve 1,2,3,4,5,4,5,6,7,8
* **INTERSECT**: Devuelve todas las filas comunes a ambas tablas
* Select a\_id from A INTERSECT select b\_id from B;🡪 devuelve 4,5
* MINUS: Devuelve todas las filas encontradas en una tabla pero no en otra
* SELECT a\_id from A MINUS select b\_id from B;🡪 devuelve 1,2,3

A veces no coinciden las tablas y tenemos que crear nosotros las tablas par que coincidan. Una forma de hacer es con TO\_CHAR(NULL), TO\_DATE(NULL), TO\_NUMBER(NULL)…. que pone NULL donde no tengamos valores para que coincidan las columnas.

* SELECT hire\_date, employee\_id, job\_id FROM employees UNION SELECT TO\_DATE(NULL), employee\_id, job\_id FROM job\_history ORDER BY employee\_id🡪 se incluye un valor NULL para cada columna que falte. Además, si se quiere ordenar las filas, se pone el ORDER BY en la última sentencia SELECT de la consulta.

## TEMA 6: SUBCONSULTAS

Las subconsultas se pueden colocar en la clausula WHERE, HAVING o FROM. Las subconsultas van entre paréntesis. Hay dos tipos de subconsultas: de una sola fila y de varias filas.

* select nombre, edad from trabajador where edad>(select edad from trabajador where nombre LIKE 'Laura Tercero');🡪Muestra todos los trabajadores con más edad que Laura Tercero.

Si una subconsulta devuelve un valor nula, la consulta externa utilizará ese valor nulo y no devolverá ninguna fila porque la comparación de cualquier valor con un valor nulo devuelve un valor nulo.

### SUBCONSULTAS DE UNA SOLA FILA

Devuelven una única fila. Utilizan comparadores de una sola fila como =,>,<,>=,<=,<>.

Las consultas externas e internas pueden obtener datos de tablas diferentes.

* SELECT last\_name, job\_id, department\_id FROM employees WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Marketing') ORDER BY job\_id🡪 devuelve quien trabaja en el departamento de markketing.

Además, más de una subconsulta puede devolver la información a la consulta externa.

* SELECT last\_name, job\_id, salary, department\_id FROM employees WHERE job\_id= (SELECT job\_id FROM employees WHERE employee\_id = 141) AND department\_id = (SELECT department\_id FROM departments WHERE location\_id = 1500);

Las funciones de grupo también se pueden utilizar en las subconsultas. Una función de grupo sin clausula GROUP BY en la subconsulta devuelve una única fila.

* select nombre, edad from trabajador where edad<(select AVG(edad) from trabajador) order by edad;🡪 devuelve las personas con menor edad que la edad media.

Las subconsultas también se pueden poner en la sentencia HAVING. Como la sentencia HAVING siempre tiene una función de grupo, las subconsulta casi siempre incluye una función de grupo también.

* SELECT department\_id, MIN(salary) FROM employees GROUP BY department\_id HAVING MIN(salary) > (SELECT MIN(salary) FROM employees WHERE department\_id = 50);🡪 devuelve los departamentos con un salario mínimo menor que el salario mínimo del departamento 50.

### SUBCONSULTAS DE VARIAS FILAS

En estas subconsultas no se pude utilizar =,<,etc. porque saltará error. Por eso utilizaremos operadores de varias filas como son ANI, IN,ALL y el operador NOT que se puede usar con cualquiera de estos operadores. En caso de que uno de los valores devueltos por una subconsulta de varias filas sea nulo, pero otros no lo sean; si usamos IN o ANY, la consulta externa devolverá filas que coinciden con los valores no nulos, pero si usamos ALL la consulta externa no devolverá nada porque ALL compara la consulta externa con cada valor devuelto por la subconsulta, incluido el nulo; y la comparación de cualquier cosa con un valor nulo es nulo.

* SELECT last\_name, employee\_id FROM employees WHERE employee\_id IN (SELECT manager\_id FROM employees);
* SELECT last\_name, employee\_id FROM employees WHERE employee\_id <= ALL (SELECT manager\_id FROM employees);

GROUP BY y HAVING también se pueden usar en subconsultas de varias filas

* SELECT department\_id, MIN(salary) FROM employees GROUP BY department\_id HAVING MIN(salary) < ANY (SELECT salary FROM employees WHERE department\_id IN (10,20)) ORDER BY department\_id; 🡪 devuelve los departamentos cuyo salario mínimo es menor que el salario de cualquier empleado que trabaja en el departamento 10 o 20.
* SUBCONSULTAS DE VARIAS COLUMNAS: pueden ser comparaciones pareadas o no pareadas
* SELECT employee\_id, manager\_id, department\_id FROM employees WHERE(manager\_id,department\_id) IN (SELECT manager\_id,department\_id FROM employees WHERE employee\_id IN (149,174)) AND employee\_id NOT IN (149,174); 🡪 esta consulta utiliza comparaciones pareadas. Aunque se han puesto las dos columnas en la misma subconsulta; se comparan por separado y por tanto es lo mismo que poner:
* SELECT employee\_id, manager\_id, department\_id FROM employees WHERE manager\_id IN (SELECT manager\_id FROM employees WHERE employee\_id IN (149,174)) AND department\_id IN (SELECT department\_id FROM employees WHERE employee\_id IN (149,174)) AND employee\_id NOT IN(149,174);🡪 donde hay una subconsulta por columna pero hace exactamente lo mismo

Algunas subconsultas pueden devolver una o varias filas dependiendo de los valores de las tablas. Es mejor hacer siempre subconsultas para varias filas por si en futuras actualizaciones de las tablas, se introduce algún valor que haga que coincida más de una fila con lo que estamos pidiendo.

* SELECT first\_name, last\_name, job\_id FROM employees WHERE job\_id = (SELECT job\_id FROM employees WHERE last\_name = 'Ernst');🡪 subconsulta de una fila
* SELECT first\_name, last\_name, job\_id FROM employees WHERE job\_id IN (SELECT job\_id FROM employees WHERE last\_name = 'Ernst');🡪 subconsulta equivalente, pero de varias filas

### Subconsultas correlacionales

* SELECT o.first\_name, o.last\_name, o.salary FROM employees o WHERE o.salary > (SELECT AVG(i.salary) FROM employees i WHERE i.department\_id = o.department\_id);🡪 devuelve de quien es el salario que es mayor que el salario medio de su departamento.
* EXIST Y NOT EXIST EN SUBCONSULTAS
* SELECT last\_name AS "Not a Manager" FROM employees emp WHERE NOT EXISTS (SELECT \* FROM employees mgr WHERE mgr.manager\_id = emp.employee\_id);🡪devuelve que empleados no son jefes. Primero vemos quienes son jefes y luego vemos quien no está en esa lista.

Cuidado con no usar NOT IN en esta consulta por el tema de los valores nulos. Las subconsultas pueden devolver 3 valores: TRUE, FALSE y UNKNOWN. Un valor NULL devolverá un valor UNKNOWN que ORACLE no puede evaluar, por lo que no lo hace.

* SELECT last\_name AS "Not a Manager" FROM employees emp WHERE emp.employee\_id NOT IN (SELECT mgr.manager\_id FROM employees mgr);🡪 devolverá que no ha encontrado resultados porque hay empleados que no tienen jefe.

Si no estamos seguros de si una subconsulta tiene valores nulos, eliminamos el valor nulo mediante IS NOT NULL en una clausula WHERE o utilizar NOT EXISTS para estar más seguros.

## TEMA 7:

Para copiar tablas usamos CREATE TABLE COPIA\_TABLA AS (SELECT \* FROM TABLAORIGINAL);

INSERT: Para agregar nuevas filas. La sintaxis es:

* INSERT INTO SEDE (ciudad, direccion, director) VALUES (‘Madrid’, ‘Puerta del Sol’, ‘Javier Perez’);

Otra forma es omitiendo los valores de las columnas, pero los valores deben coincidir con el orden por defecto

* INSERT INTO SEDE ) VALUES (‘Madrid’, ‘Puerta del Sol’, ‘Javier Perez’);

Se pueden insertar valores nulos, pero solo en aquellas columnas que lo permitan.

* INSERT INTO DONACION VALUES(1, ‘23/04/20’,’ Valencia’,’ 06963862V’,’ 7635DKZ’, );🡪 El último valor es nulo.

Para agregar explícitamente valores nulos poner NULL

* Inserción de valores especiales como SYSDATE(fecha y hora actual) y USER(Usuario): se ponen tal cual en la lista de VALUES.
* INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (304,'Test',**USER**, 't\_user', 4159982010, **SYSDATE**, 'ST\_CLERK',2500);
* Inserción de valores de fecha específicos: el modelo de formato especifico para datos de fecha es DD-MMM-YYYY. Si queremos insertar una fila con un formato de fecha distinto al por defecto podemos utilizar la función TO\_DATE(la función inversa es el TO\_CHAR):
* INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (301,'Katie','Hernandez', 'khernandez','8586667641', **TO\_DATE('July 8, 2015', 'Month fmdd, yyyy'),** 'MK\_REP',4200);
* INSERT INTO copy\_employees (employee\_id, first\_name, last\_name, email, phone\_number, hire\_date, job\_id, salary) VALUES (303,'Angelina','Wright', 'awright','4159982010'**, TO\_DATE('July 10, 2015 17:20', 'Month fmdd, yyyy HH24:MI')**, 'MK\_REP', 3600);🡪 Igual que la anterior pero aquí, además, ponemos también la hora.
* Uso de una subconsulta para copiar filas. Para introducir o copiar más de una fila en una tabla. Por ejemplo, si queremos copiar 100 filas de una tabla a otra de una sola vez.
* Insert into copiaDonacion select \* from donacionOriginal where destino like ‘Valencia’; 🡪 inserta en copiaDonacion todas las donaciones de la tabla original cuyo destino es Valencia.
* INSERT INTO sales\_reps(id, name, salary, commission\_pct) SELECT employee\_id, last\_name, salary, commission\_pct FROM employees WHERE job\_id LIKE '%REP%';

Se pueden elegir las columnas que se quieren copiar para que coincidan con la tabla destino.

### Actualizar y eliminar filas

* UPDATE

Necesita 4 valores:

* Nombre de la tabla
* Columnas que se van a modificar
* Nuevos valores para las columnas
* Condición que identifique las filas que se deben actualizar en la clausula WHERE. Si se omite, se actualizará en todas las filas de la tabla

Por ejemplo:

* UPDATE copiadonacion set destino= 'Leon', fecha= SYSDATE where id\_donacion=1;
* update copiadonacion set destino= 'Leon', fecha= SYSDATE;🡪 cambiará los valores destino y fecha en todas las filas de la tabla.

Se pueden actualizar los valores utilizando los resultados de una subconsulta de una sola fila.

* UPDATE copy\_employees SET salary = (SELECT salary FROM copy\_employees WHERE employee\_id = 100) WHERE employee\_id = 101;🡪 actualiza una columna con el valor de una subconsulta de una sola fila.
* UPDATE copy\_employees SET salary = (SELECT salary FROM copy\_employees WHERE employee\_id = 205), job\_id = (SELECT job\_id FROM copy\_employees WHERE employee\_id = 205) WHERE employee\_id = 206;🡪 para actualizar varias columnas es posible hacer varias subconsultas de una sola fila, una para cada columna.

También se pueden hacer subconsultas correlacionadas.

* UPDATE copy\_employees e SET e.department\_name = (SELECT d.department\_name FROM departments d WHERE e.department\_id = d.department\_id);
* DELETE:

Necesita dos valores:

* Nombre de la tabla
* Condición que identifique la fila que queremos eliminar

Si se omite la clausula where se borran todas las filas de la tabla

Por ejemplo:

* DELETE from copy\_employees WHERE employee\_id = 303;

Es posible utilizar subconsultas de mas de una fila para identificar las líneas que hay que eliminar.

* DELETE FROM copy\_employees e WHERE e.manager\_id IN (SELECT d.manager\_id FROM employees d HAVING count (d.department\_id) < 2 GROUP BY d.manager\_id);🡪 elimina todos los empleados que trabajan para un gerente que administra menos de 2 empleados.

## TEMA 8: TABLAS

### Creación de tablas

La sintaxis es:

CREATE TABLE table

(column data type [DEFAULT expression],

(column data type [DEFAULT expression],

(……[ ] );

Donde default expresión especifica un valor por defecto si se omite un valor en la sentencia INSERT.

### Uso de tipo de datos

Para valores de carácter:

* CHAR (tamaño fijo, máximo 2000 caracteres) es el menos recomendable
* VARCHAR2 (tamaño variable, máximo 4000 caracteres)
* CLOB (tamaño variable, máximo 128 terabytes)

Para valores numéricos:

* NUMBER (tamaño variable, precisión máxima de 38 digitos)

Para valores binarios (p. ej. , multimedia: JPG,WAV…):

* RAW (tamaño variable, máximo de 2000 bytes)
* BLOB (tamaño variable, máximo de 128 terabytes)

Para valores de fecha:

* DATE: almacena siglos hasta segundos enteros, pero no permite fracciones de un segundo. Ej: “21-Aug-2021 17:23:50” es válido, pero “21-Aug-2021 17:23:50.255” no lo es
* INTERVAL
* TIMESTAMP: extensión de date pero acepta fracciones de un segundo

Ej. INSERT INTO time\_exc VALUES (SYSTIMESTAMP);

Ej. Create table time\_exc (exact\_time TIMESTAMP);

IMPORTANTE saber a que zona horaria pertenece la hora; para ello se utiliza TIMESTAMP WITH TIME ZONE

Ej. CREATE TABLE time\_ex (time TIMESTAMP WITH TIME ZONE);

Ej. INSERT INTO time\_ex VALUES(“10-Jun-2021 10:20:30.123 AM +2.00”); 🡪 2 horas por encima de UTC

Con TIMESTMAP WITH LOCAL TIME ZONE es igual pero se cambia automáticamente a la zona horaria del usuario

### Modificación de una tabla

* **ALTER TABLE**: Se usan para agregar una columna, modificar una columna existente, definir un valor default para un columna y borrar una columna. Una nueva columna se pondrá al final y todos los valores de todas las filas existentes se pondrán a NULL.

Para agregar una columna:

* ALTER TABLE my\_cd\_collection ADD (release\_date DATE DEFAULT SYSDATE);🡪 se añade con un valor por defecto
* ALTER TABLE my\_friends ADD (favorite\_game VARCHAR2(30))🡪 se añade sin un valor por defecto; únicamente diciendo cual es el tipo de dato

Para modificar una columna: podrá aumentar el ancho y precisión de una columna de números y de caracteres y solo podrá reducir el ancho de una columna de números si no hay filas o son nulos; solo podrá reducir el ancho de una columna de caracteres hasta el mayor valor existente en la columna. Podrá cambiar el tipo de datos solo si hay valores nulos en la columna

* CREATE TABLE mod\_emp (last\_name VARCHAR2(20), salary NUMBER(8,2))
* ALTER TABLE mod\_emp MODIFY (last\_name VARCHAR2(30));
* ALTER TABLE mod\_emp MODIFY (salary NUMBER(8,2) DEFAULT 50);🡪 podrá hacerse porque solo cambia el valor por defecto
* ALTER TABLE mod\_emp MODIFY (last\_name VARCHAR2(10));🡪 solo estaría permitido si hubiera solo valores nulos o no hubiera filas

Para borrar una columna: solo se puede borrar una columna cada vez. No se pueden borrar todas las columnas de una tabla; debe permanecer al menos una.

* ALTER TABLE my\_cd\_collection DROP COLUMN release\_date;

Para eliminar una tabla completamente utilizamos **DROP TABLE nombreTabla;**

### Restricciones

Las restricciones que hacen referencia a mas de una columna se deben especificar a nivel de tabla. La restricción NOT NULL se debe especificar a nivel de columna. Las restricciones UNIQUE, PRIMARY KEY, FOREIGN KEY y CHECK se pueden especificar en el nivel de columna o en el de tabla. Si se utiliza la palabra CONSTRAINT se debe asignar un nombre a la restricción; en caso contrario no hace falta.

* Ej. CREATE TABLE clients

( client\_number NUMBER(6),

first\_name VARCHAR2(20),

last\_name VARCHAR2(20),

phone VARCHAR2(20) CONSTRAINT phone\_email\_uk UNIQUE(email,phone),

email VARCHAR2(10) CONSTRAINT NOT NULL,

CONSTRAINT emailclients\_email NOT NULL,

CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number));

* UNIQUE: Los valores de esa columna o esas columnas deben ser únicos. Cuando se combina varias columnas en una clave UNIQUE se dice que es una clave única compuesta.
* CONSTRAINT clients\_phone\_email\_uk UNIQUE(email,phone)🡪 la combinación (email,pone) debe ser única en la tabla.
* PRIMARY KEY: regla de que los valores de una columna o una combinación de columnas deben identificar a una fila de forma única. Se puede definir o en el nivel de columna o de tabla pero si se crea una clave primaria compuesta, se debe definir en el nivel de tabla.
* CREATE TABLE clients

(client\_number NUMBER(4) CONSTRAINT clients\_client\_num\_pk PRIMARY KEY,

first\_name VARCHAR2(14),

last\_name VARCHAR2(13));

* CREATE TABLE clients

(client\_number NUMBER(4),

first\_name VARCHAR2(14),

last\_name VARCHAR2(13),

CONSTRAINT clients\_client\_num\_pk PRIMARY KEY (client\_number));

* FOREIGN KEY: Designan una columna o una combinación de columnas como clave ajena. Una clave ajena enlaza con la clave primaria de otra tabla.
* CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

last\_name VARCHAR2(25),

department\_id NUMBER(4,0) CONSTRAINT c\_emps\_dept\_id\_fk REFERENCES departments(department\_id),

email VARCHAR2(25));🡪 definida a nivel de columna

* CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

email VARCHAR2(25),

CONSTRAINT c\_emps\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES departments(department\_id));🡪 definida a nivel de tabla

* ON DELETE CASCADE: al definir la clave ajena permite la supresión de todas las filas dependientes en la tabla al eliminar una fila de la tabla principal. Si no se usa esto, las filas a las que hace referencia en la tabla principal no se pueden eliminar. La restricción FOREIGN KEY incluye el permiso ON DELETE CASCADE. Si no se usa esta opción, devolverá error al intentar borrar la fila de la tabla principal.
* CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES copy\_departments(department\_id) ON DELETE CASCADE);🡪 no devolverá fallo

* ON DELETE SET NULL: Igual que la anterior, pero en lugar de eliminar las filas secundarias, estas se ponen con valores nulos.
* CREATE TABLE copy\_employees

(employee\_id NUMBER(6,0) CONSTRAINT copy\_emp\_pk PRIMARY KEY,

last\_name VARCHAR2(25),

department\_id NUMBER(4,0),

CONSTRAINT cdept\_dept\_id\_fk FOREIGN KEY (department\_id) REFERENCES copy\_departments(department\_id) ON DELETE SET NULL);

* CHECK: Define explícitamente una condición que se debe cumplir en todas las filas.
* CREATE TABLE copy\_job\_history

(employee\_id NUMBER(6,0),

start\_date DATE,

end\_date DATE,

department\_id NUMBER(4,0),

CONSTRAINT cjhist\_emp\_id\_st\_date\_pk PRIMARY KEY(employee\_id, start\_date), CONSTRAINT cjhist\_end\_ck **CHECK** (end\_date > start\_date));🡪 como hace referencia a dos columnas; se debe definir en el nivel de tabla. Si no fuera asi, se podría definir en nivel de columna.

## TEMA 9: VISTAS

Tenemos dos tipos de vistas: simples(subconsultas de una tabla) y complejas(subconsultas de 2 o más tablas). La sintaxis es CREATE VIEW <restricción> nombrevista AS <funcion>;

* SIMPLES:
* Create view OR REPLACE “personas jóvenes” AS SELECT nombre,edad,sede FROM TRABAJADOR WHERE edad<=23;🡪 después solo habría que poner select \* from “personas jóvenes” y se ejecutaría esa consulta.

Se pueden poner alias a las columnas de dos formas:

* Create view or replace “personas jóvenes”(“name”,”age”,”place”) as select nombre,edad,sede from trabajador where edad<=23
* Create view or replace “personas jóvenes” as select nombre as “name”, edad as “age”, sede as “place” from trabajador where edad<=23;

Si ponemos alias; a la hora de ejecutar la vista debemos utilizar siempre esos alias.

* COMPLEJAS: Pueden contener funciones de grupo y uniones
* CREATE OR REPLACE VIEW view\_euro\_countries ("ID", "Country", "Capitol City", "Region") AS SELECT c.country\_id, c.country\_name, c.capitol, r.region\_name FROM wf\_countries c JOIN wf\_world\_regions r USING (region\_id) WHERE location LIKE '%Europe';🡪 select \* from view\_euro\_countries
* CREATE OR REPLACE VIEW view\_high\_pop ("Region ID", "Highest population") AS SELECT region\_id, MAX(population) FROM wf\_countries GROUP BY region\_id;

Para modificar una vista existente sin tener que eliminarla se puede utilizar la opción OR REPLACE.



